

INFO-F-302 : Logique informatique

Projet : Le Jeu ABCPath et Utilisation d'un Solveur SAT

L'objectif de ce projet est de modéliser plusieurs problèmes en logique propositionnelle autour du jeu ABCPath, et de résoudre ces problèmes à l'aide du solveur SAT MiniSat.

1 Le Jeu ABCPath

Une instance du jeu ABCPath est donnée par une grille 7x7 :

```

B C D W J N F
R A - - - - X
S - - - - - V
U - - - - - O
G - - - - - I
M - - - - - E
T Y H K Q P L

```

Nous appellerons *contour* la partie extérieure de la grille, c'est-à-dire celle où apparaissent les lettres B,C,D,W,J,N,F,X,V,etc... Nous appellerons *intérieur* le reste de la grille, c'est-à-dire le carré 5x5 se trouvant au milieu.

L'objectif est de remplir l'intérieur de cette grille avec les lettres A jusque Y avec les contraintes suivantes (la position du A est toujours donnée) :

1. chaque lettre n'apparaît qu'une seule fois à l'intérieur ;
2. pour toute lettre α qui n'est pas Y, la lettre suivante dans l'ordre alphabétique apparaît dans le voisinage direct de α . Autrement dit, si (i, j) est la position de α , la lettre suivante se trouve en position $(i + d, j + e)$ pour un certain couple $(d, e) \in \{-1, 0, 1\} \times \{-1, 0, 1\}$;
3. pour chaque lettre du contour, une lettre de la colonne / ligne / diagonale correspondante doit apparaître dans l'intérieur de la grille. Par exemple, le C doit apparaître dans la première colonne de l'intérieur, le S dans la deuxième ligne de l'intérieur, le K dans la troisième colonne dans l'intérieur, le B dans la diagonale de l'intérieur qui commence en haut à gauche, etc ...

Voici une solution pour cette grille :

```

B C D W J N F
R A X W Q R X
S Y B V S P V
U C H T U O O
G G D I L N I
M F E K J M E
T Y H K Q P L

```

Vous pouvez vous entraîner à résoudre des grilles à l'adresse suivante :

<http://www.brainbashers.com/abcpath.asp>

Dans ce projet, il vous sera demandé de résoudre des grilles à l'aide d'un solveur SAT, de tester l'unicité d'une solution, de générer des grilles, avec au moins deux solutions ou avec solution unique, tout cela pour des grilles de tailles variables : 7x7, 8x8 ou 9x9.

On peut en effet généraliser le problème à des grilles de taille plus grande, on aura alors besoin de plus de lettres. Nous utiliserons les lettres minuscules a,b,c,etc ... qui seront triées dans l'ordre suivant : A,B,...,Y,Z,a,b,...,z. Par exemple, pour une grille 8x8, la grille intérieure aura une taille 6x6, donc on prendra les 36 lettres dans l'intervalle [A,j], et les contraintes de contour seront elles aussi exprimées avec des lettres de l'intervalle [A,j]. Voici par exemple une grille 8x8 :

```
F H d K L M S O
J - - - - A E
D - - - - B
G - - - - C
R - - - - P
Y - - - - U
e - - - - g
a I b Z X W T Q
```

et une solution :

```
F H d K L M S O
J I J E L M A E
D H F K D N B B
G G b j O C S C
R c i a P R T P
Y h d Z Y Q U U
e g f e X W V g
a I b Z X W T Q
```

2 Le Solveur SAT MiniSat

Le solveur SAT MiniSat prend en entrée une formule de logique propositionnelle en **forme normale conjonctive**, en teste la satisfaisabilité et retourne une valuation qui satisfait la formule dans le cas où elle est satisfaisable. Dans ce projet, nous vous laissons le choix de l'utilisation de MiniSat : vous pouvez soit programmer dans le langage de votre choix et faire des appels à MiniSat via son système d'entrées/sorties, soit programmer en C++ et utiliser directement la bibliothèque C++ MiniSat que nous vous fournissons. Dans le premier cas, vous devrez installer MiniSat vous-même à partir de l'adresse suivante :

<http://minisat.se/MiniSat.html>.

MiniSat prend des fichiers au format Dimacs expliqué ici :

<http://logic.pdmi.ras.ru/~basolver/dimacs.html>

Si vous utilisez la bibliothèque MiniSat que nous vous fournissons, vous pouvez repartir du code utilisé pour la résolution du problème Sudoku, disponible à l'adresse suivante :

<http://www.ulb.ac.be/di/info-f-302/index.html>

Dans cette bibliothèque, chaque variable propositionnelle est représentée par un entier. Pour créer une clause, il faut créer une liste de littéraux. Voici un exemple de code qui crée la formule $(p_0 \vee \neg p_1) \wedge (\neg p_0 \vee p_2)$ et affiche une valuation si la formule est satisfaisable. La variable p_i est représentée par l'entier i pour tout $i \in \{0, 1, 2\}$.

```
#include "Solver.hpp"

// déclaration d'un solveur MiniSat
Solver s;

// ajout de trois variables
for (int i = 0 ; i <= 2 ; i++) { s.newVar() ; }
```

```

// déclaration d'un vecteur de littéraux
vec<Lit> lits;

// clause p0 or (not p1)
lits.push(Lit(0)) ; // ajout du littéral 0
lits.push(~Lit(1)) ; // ajout du littéral (non 1)
s.addClause(lits) ; // ajout de la clause

// clause (not p0) or p2
lits.clear(); // réinitialisation du vecteur de clauses
lits.push(~Lit(0)) ; // ajout du littéral (non 0)
lits.push(Lit(2)) ; // ajout du littéral 2
s.addClause(lits) ; // ajout de la clause

if (s.solve()) // résolution par MiniSat
{ // la formule est satisfaisable

    cout << "La formule est satisfaisable avec la valuation où\n" ;
    for (int i = 0 ; i <= 2 ; i++) { // récupération de la valuation
        if (s.model[i] == l_True)
            cout << "la variable " << i << " est mise à vraie\n";
        else
            cout << "la variable " << i << " est mise à faux\n";
    }
}
else
    cout << "La formule n'est pas satisfaisable\n" ;

```

Si votre clause ne contient qu'un seul littéral, vous pouvez utiliser la méthode `addUnit()`, par exemple, `addUnit(Lit(0))`. Si elle ne contient que deux littéraux, vous pouvez utiliser par exemple `addBinary(Lit(0), Lit(2))`.

3 Questions

Dans toutes ces questions, vous pouvez vous contenter de ne considérer que des grilles de taille 5x5. Si votre programme prend en plus la taille de la grille comme un paramètre défini par l'utilisateur, il vous sera attribué un bonus. Toutefois, vous pouvez vous limiter à des grilles de taille 9x9 au plus. Attention, il se peut que les temps de calcul soient longs, jusqu'à plusieurs minutes pour les grandes grilles, soyez patients.

Question 1 (Résolution d'une grille) *Ecrire une procédure qui prend en entrée une grille au format texte de la section 1 et qui résout cette grille en utilisant **MiniSat**. Votre procédure devra donner la solution sous format texte comme dans la section 1. Votre procédure devra générer une instance du problème SAT telle que les valuations qui satisfont la formule (s'il en existe) représentent des solutions de la grille. Expliquer comment vous coder le problème comme une instance du problème SAT (représentation de la grille, des lettres, des variables, formules générées, éventuellement expliquer comment vous mettez sous forme normale conjonctive). A l'adresse <http://www.ulb.ac.be/di/info-f-302/index.html> vous trouverez un ensemble de grilles 7x7, 8x8 et 9x9 pour tester votre procédure.*

Question 2 (Génération de grilles) *Ecrire une procédure qui génère une grille (c'est-à-dire un contour et le placement de la lettre A de départ) telle que la grille générée a au moins une solution. Pour cette question, la contrainte est la suivante : une même lettre ne peut pas apparaître plus d'une fois sur le contour, et le A n'apparaît pas sur le contour (cette contrainte serait en effet inutile car la position du A est donnée au départ). Pour cette question, il ne s'agit pas de générer aléatoirement des grilles et de tester si elles ont une solution. Comme pour la question 1, on vous demande de générer une instance du problème SAT telle que toute valuation qui satisfait l'instance représente une grille qui possède au moins une solution. Comme pour la question 1, expliquez votre codage du problème en logique propositionnelle. Vous pourrez tester votre procédure en utilisant la procédure de la question 1 pour résoudre les grilles que vous générez.*

Etendre votre procédure pour qu'elle génère n grilles différentes (n étant donné en paramètre par l'utilisateur). Expliquez comment vous procéder.

Pour accélérer votre programme lors de la génération de n grilles, vous pouvez utiliser *l'incrémentalité* de MiniSat. Par exemple, si vous ajoutez des clauses supplémentaires à une formule, au lieu de déclarer un nouveau solveur, vous pouvez relancer le solveur précédent avec la nouvelle formule. MiniSat réutilise en effet les résultats précédents pour la résolution de la nouvelle formule. Par exemple, le programme suivant :

```
Solver s;  
createClauses1(s);  
s.solve();  
  
Solver s';  
createClauses1(s');  
createClauses2(s');  
s'.solve();
```

où les procédures `createClauses1` et `createClauses2` génèrent un ensemble de clauses et l'ajoute au solveur, peut être optimisé en :

```
Solver s;  
createClauses1(s);  
s.solve();  
  
createClauses2(s);  
s.solve();
```

Notez que cette fonctionnalité ne peut être utilisée que si vous utilisez les sources de MiniSat.

Question 3 (Unicité de la solution) *Ecrire une procédure qui résout une grille comme dans la question 1 et qui teste si la solution est unique via une instance de SAT. Si la solution n'est pas unique, votre procédure devra sortir une deuxième solution. Attention, n'oubliez pas que la lettre A est toujours donnée au départ et ne peut donc pas être déplacée. Expliquez comment vous procéder, en particulier, expliquez quelles formules vous générez. Vous pouvez tester votre procédure sur les exemples qu'on vous a donnés, dans les répertoires **unique** pour les grilles avec solution unique, et les répertoires **notunique** pour les grilles avec au moins deux solutions, pour des grilles de taille 7x7 et 8x8.*

Pour la question suivante, on pourra se limiter à des grilles de taille 8x8 au plus.

Question 4 (Bonus 1 : Génération de grilles avec au moins deux solutions) *Ecrire une procédure qui génère n grilles différentes (n étant donné en entrée) telle que chaque grille a au moins deux solutions. On ne vous demande **pas** ici de procéder en deux étapes : générer une grille et tester si la solution est unique, recommencer sinon, jusqu'à temps que n grilles soient trouvées. On vous demande plutôt de procéder en une seule étape : générer une seule instance de SAT telle que les valuations qui satisfont cette instance représentent des grilles avec au moins deux solutions.*

Pour la question précédente, vous aurez peut-être besoin d'utiliser l'astuce suivante pour mettre une formule sous forme normale conjonctive, en ajoutant des variables. Prenons par exemple la formule suivante :

$$\bigwedge_{i=1}^{n_1} \bigvee_{j=1}^{n_2} \bigwedge_{k=1}^{n_3} l_{i,j,k}$$

où pour tout $i \in \{1, \dots, n_1\}$, $j \in \{1, \dots, n_2\}$, $k \in \{1, \dots, n_3\}$, $l_{i,j,k}$ est un littéral.

Pour mettre cette formule en forme normale conjonctive, vous pouvez distribuer la disjonction au risque de créer une formule exponentiellement plus grande, ou utiliser l'astuce suivante : pour

tout i, j , ajouter la variable $y_{i,j}$ et transformer la formule comme ceci :

$$\left(\bigwedge_{i=1}^{n_1} \bigvee_{j=1}^{n_2} y_{i,j}\right) \wedge \left(\bigwedge_{i=1}^{n_1} \bigwedge_{j=1}^{n_2} (y_{i,j} \leftrightarrow \bigwedge_{k=1}^{n_3} l_{i,j,k})\right)$$

Cette formule peut facilement être mise sous forme normale conjonctive :

$$\left(\bigwedge_{i=1}^{n_1} \bigvee_{j=1}^{n_2} y_{i,j}\right) \wedge \left(\bigwedge_{i=1}^{n_1} \bigwedge_{j=1}^{n_2} \bigwedge_{k=1}^{n_3} (\neg y_{i,j} \vee l_{i,j,k})\right) \wedge \left(\bigwedge_{i=1}^{n_1} \bigwedge_{j=1}^{n_2} y_{i,j} \vee \left(\bigvee_{k=1}^{n_3} \neg l_{i,j,k}\right)\right)$$

Il est donc possible, en introduisant de nouvelles variables, de mettre en forme normale conjonctive une formule en évitant l'explosion combinatoire. Si on applique cette astuce à la formule suivante $(a \wedge b \vee a \wedge c) \wedge (\neg a \wedge b \vee d)$, on obtient en première étape la formule :

$$(y_1 \vee y_2) \wedge (y_3 \vee y_4) \wedge (y_1 \leftrightarrow a \wedge b) \wedge (y_2 \leftrightarrow a \wedge c) \wedge (y_3 \leftrightarrow \neg a \wedge b) \wedge (y_4 \leftrightarrow \neg a \wedge d)$$

Chaque double équivalence peut facilement être mise sous forme normale conjonctive, et la formule devient :

$$\begin{aligned} & (y_1 \vee y_2) \wedge (y_3 \vee y_4) \wedge \\ & \wedge (\neg y_1 \vee a) \wedge (\neg y_1 \vee b) \wedge (y_1 \vee \neg a \vee \neg b) \wedge \\ & \wedge (\neg y_2 \vee a) \wedge (\neg y_2 \vee c) \wedge (y_2 \vee \neg a \vee \neg c) \wedge \\ & \wedge (\neg y_3 \vee \neg a) \wedge (\neg y_3 \vee b) \wedge (y_3 \vee a \vee \neg b) \wedge \\ & \wedge (y_4 \vee \neg a) \wedge (\neg y_4 \vee d) \end{aligned}$$

Toute valuation qui satisfait cette dernière formule satisfait aussi la formule d'origine. Réciproquement, toute valuation ν de la formule d'origine (donc des variables a, b, c, d) peut être transformée en une valuation β qui satisfait la nouvelle formule de la manière suivante : $\beta(a) = \nu(a)$, $\beta(b) = \nu(b)$, $\beta(c) = \nu(c)$, $\beta(d) = \nu(d)$ et $\beta(y_1) = \nu(a) \wedge \nu(b)$, $\beta(y_2) = \nu(a) \wedge \nu(c)$, $\beta(y_3) = \neg \nu(a) \wedge \nu(b)$, et $\beta(y_4) = \neg \nu(a) \wedge \nu(d)$.

Pour la question suivante on pourra se limiter aux grilles de taille 7x7, mais si vous êtes capables d'y répondre pour des grilles plus grande, c'est un plus.

Question 5 (Bonus 2 : Génération de grilles avec solution unique) *En vous aidant du solveur MiniSat, écrire un algorithme qui génère n grilles telles que pour chaque grille, la solution est unique. Pour cette question il n'est pas demandé de résoudre le problème directement par n appels à MiniSat, mais il faut vous aider de MiniSat. Conseil : observez les raisons pour lesquelles une solution n'est pas unique, et essayez d'interdire certains motifs lors de la génération des grilles.*

Modalités

Le projet se fait en **binôme**, il est à rendre au bureau de Maryka Peetroons pour le **8 Mai 16h**.

Il doit comprendre un rapport papier qui répond aux questions et explique la manière dont vous codez les différents problèmes comme des instances de SAT. Il vous est demandé de rendre les sources de votre programme, avec un manuel d'installation / utilisation. Pour la génération de grilles, on vous demande de donner dans un fichier 5 grilles générées pour chaque taille, 7x7, 8x8 et 9x9. Les grilles seront au format texte comme dans la section 1, séparées par une ligne vide. Vous devez donc rendre trois fichiers pour la question 2 (chacun avec 5 grilles), deux fichiers pour la question 4 (car on se limite aux grilles de taille 8x8 au plus), et un fichier pour la question 5 (on se limite aux grilles de taille 7x7). Le tout (rapport, grilles générées et sources) devra être rendu dans un dossier compressé dont le nom portera les **deux** noms de famille du binôme, et envoyé à l'adresse efiliot@ulb.ac.be avec pour objet **Projet Logique : nom de famille1-nom de famille2**. Le fichier est à envoyer pour la même date que la version papier.